REMARKS

Claims 1-26 are pending in the present application. Claim 1 has been amended to refer back to limitations in the preamble, to ensure that these limitations are considered in this claim. Reconsideration of the claims is respectfully requested.

Amendments were made to the specification, as required by the Examiner, to add information that was not available at the time of filing and to remove a website reference. No new matter has been added by any of the amendments to the specification.

I. 35 U.S.C. § 102, Anticipation

The examiner has rejected claims 6-11, 17-22, and 25 under 35 U.S.C. § 102(e) as anticipated by Meltzer et al. (6,226,675). This rejection is respectfully traversed.

Representative claim 6 recites:

> *6. A method of generating a markup language file, the method comprising the computer-implemented steps of:*
> *executing an application program;*
> *parsing a document type definition file for a markup language;*
> *selecting an element defined in the document type definition file based on a routine called by the application program; and*
> *writing the selected element to a markup language file.*

This claim is describing some of the steps of a dynamic translation program. As the application program of the first step is executing, an element in the DTD[1] file is selected, based on a routine called by the executing program. This element, which represents a markup language equivalent of the routine called by the application program, is saved to the translation file.

In his rejection of claim 6, the Examiner cites Meltzer as disclosing:

> *A method ... for generating a markup language file, comprising:*
> *executing an application program (Meltzer on col. 23, lines 17-60: teaches JAVA (application program));*
> *parsing a document type definition file for a markup language (Meltzer on col. 23, lines 38-60: teaches parsing XML DTD);*
> *selecting an element defined in the document type definition file based on a routine called by the application program (Meltzer on col. 23, lines 38-60): teaches element retrieved from XML DTD and col. 23, lines 17-60: teaches JAVA (application program); and*

---

[1] document type definition

writing the selected element to a markup language file (Meltzer on col. 23, lines 38-60: teaches producing an output by received XML element).

It is submitted that, while there is a superficial resemblance between the cited method in Meltzer and the method of Claim 6, there are also basic differences between the two methods that are ignored in the discussion of Meltzer above. In order to clarify these differences, let us look first at the problems Meltzer is directed toward.

Meltzer suggests that "*to open commercial transactions of the Internet, standardization of architectural frameworks is desired.*"[2] There are many proprietary frameworks designed to handle Internet commerce, yet "*a consumer or business using one framework is unable to execute transactions on a different framework. This limits the growth of electronic commerce systems.*"[3] As a solution to this problem, the invention is described as allowing "*companies* [to] *exchange information and services using self-defining, machine readable documents, such as XML ... based documents, that can be easily understood amongst the partners.*"[4] Thus, Meltzer is directed, not to translating a program into XML, as is the instant application, but to translating a document containing data to and from XML and another architecture. To this end, Meltzer discloses a node of a network (e.g., the Internet) as it communicates with the outside world, illustrated in his Figure 3, below.

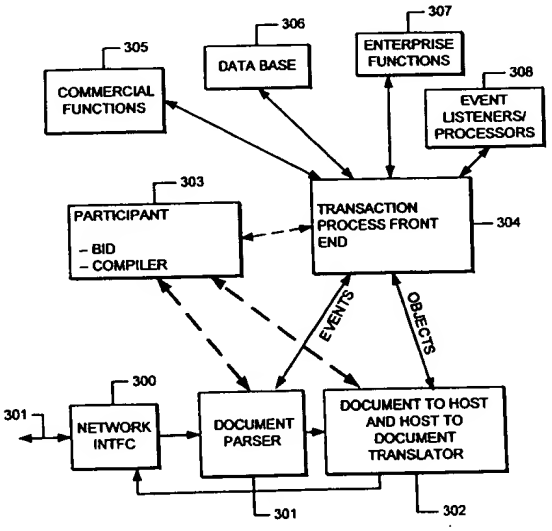

FIG. 3

[2] Meltzer *et al.*, col. 1, line 59-60
[3] Meltzer *et al.*, col. 2, lines 6-9
[4] Meltzer *et al.*, col. 2, lines 32-38

Meltzer notes that this node

*"includes a network interface **300** which is coupled to a communication network on port **301**. The network interface is coupled to a document parser **301**. The parser **301** supplies the logical structures from an incoming document to the translator module **302**, which provides for translating the incoming document into a form usable by the host transaction system, and vice versa translating the output of host processes into the format of a document which matches the output document form in the business interface definition for transmission to a destination. The parser **301** and translator **302** are responsive to the business interface definition stored in the participant module **303**."*

Let us now look at the portions of Meltzer that the Examiner has read on Claim 6:

*"Furthermore, according to the present invention the application that the listeners run need not be native XML functions, or native functions which match the format of the incoming document. Rather, these listeners may be JAVA functions, if the transaction process front end **304** is a JAVA interface, or may be functions which run according to a unique transaction processing architecture. In these cases, the objects would be transformed into the format required by the receiving applications. When the application of the listener finishes, its output is then transformed back into the format of a document as specified by the business interface definition in the module **303**. Thus, the translator **302** is coupled to the network interface **300** directly for supplying the composed documents as outputs.*

The listeners coupled to the transaction processing front end may include listeners for input documents, listeners for specific elements of the input documents, and listeners for attributes stored in particular elements of the input document. This enables diverse and flexible implementations of transaction processes at the participant nodes for filtering and responding to incoming documents.

*FIG. **4** illustrates a process of receiving and processing an incoming document for the system of FIG. **3**. Thus, the process begins by receiving a document at the network interface (step **400**). The parser identifies the document type (401) in response to the business interface definition. Using the business interface definition, which stores a DTD for the document in the XML format, the document is parsed (step **402**). Next, the elements and attributes of the document are translated into the format of the host (step **403**). In this example, the XML logic structures are translated into JAVA objects which carry the data of the XML element as well as methods associated with the data such as get and set functions. Next, the host objects are transferred to the*

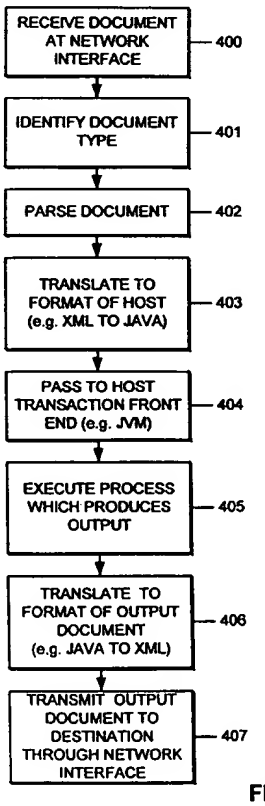

FIG. 4

*host transaction processing front end (step **404**). These objects are routed to processes in response to the events indicated by the parser and the translator. The processes which receive the elements of the document are executed and produce an output (step **405**). The output is translated to the format of an output document as defined by the business interface definition (step **406**). In this example, the translation proceeds from the form of a JAVA object to that of an AML document. Finally, the output document is transmitted to its destination through the network interface (step **407**)."*

All of the above strengthens the assertion that Meltzer is translating documents, not applications. If we try to fit Meltzer's translation to the claim language of Claim 6, the business interface definitions (BID) of Meltzer's Figure 3 might be read on the claimed "*document type definition*", but there is not application program to be executed and translated, only a document to be translated. Not only is there no application program, but there is no routine called by the application program to which we can associate an element from the document type definition file. Since one cannot select an element that corresponds to a routine, neither can one write the selected element to a markup file. Thus, none of the claims limitations appear to be met by Meltzer.

The Examiner has read JAVA on the limitation for "*executing an application program*". It is true that JAVA is mentioned in the section above. However, if JAVA is to be read on Claim 6 as the application program being translated, the Examiner should be able to point to a routine called by JAVA for which a corresponding "*element defined in the document type definition file*" is selected. Instead, the Examiner is attempting to pull in divergent elements of Meltzer and combine them in a manner that they are not meant to be combined. It is submitted that this combination would not be done by one of ordinary skill in the art.

As has been seen above, this rejection does not hold up under closer scrutiny. This rejection is overcome for Claim 6. Claims 12 and 23 are rejected for the same reasons as Claim 6 and are thus overcome for the same reason.

Since Claims 7-11 and 13-20 depend from Claims 6 and 12, the same distinctions between Meltzer and the claimed invention in Claim 1 apply for these claims. Additionally, many of these dependent claims claim other additional combinations of features not suggested by the reference.

For instance, Claim 7 recites that "the element comprises an attribute list corresponding to parameters for the routine."

Claim 8 recites that "the selected element written to the markup language file comprises an attribute list corresponding to values for the parameters passed to the routine."

Claim 18 recites that "the element comprises an attribute list of parameters for the routine."

Claim 19 recites that "the selected element written to the markup language file comprises an attribute list of values for the parameters passed to the routine."

Claim 20 recites that "the application program is written in Java programming language."

Consequently, it is respectfully urged that the rejection of claims 7-11 and 17-22 have been overcome.

Therefore, the rejection of claims 6-11, 17-22, and 25 under 35 U.S.C. § 102 has been overcome.

Furthermore, Meltzer does not teach, suggest, or give any incentive to make the needed changes to reach the presently claimed invention. While both Meltzer and the present application are concerned with translations, they are not translating the same things. Meltzer, in the claim language, is performing a dynamic translation of an application as it executes; this does not correspond to simply translating an application. One of ordinary skill in the art would not be led to modify Meltzer to reach the present invention when the reference is examined as a whole.

V. 35 U.S.C. § 103, Obviousness

The examiner has rejected claims 1-5, 12-16, 23-24, and 26 under 35 U.S.C. §103(a) as being unpatentable over Meltzer et al. (6,226,675) in view of Day et al. (5,953,526). This rejection is respectfully traversed.

Representative Claim 1 reads:

1. (Amended) A method of processing a source code statement written in a programming language, the method comprising the computer-implemented steps of:

parsing a document type definition file for a markup language;

parsing said source code statement from a source code file;

selecting an element defined in the document type definition file based on an association between the element and an identifier of a routine in said source code statement; and

writing the selected element to a markup language file.

This claim is similar to Claim 6 discussed above, except that the translation process is static, performed on source code, rather than dynamic, occurring as the code is executed as an application. As the translation process proceeds through the source code file, for each routine identified in the source code, a corresponding element is selected from the DTD and written to the translation, or markup language, file.

In his rejection, the Examiner notes that "*Meltzer does not explicitly disclose "parsing a source code statement from a source code file". However, Day on col. 7, lines 24-50 and col. 8, lines 12-45: teaches parsing JAVA file to look for package statement.*"

It is first noted that it can be tricky combining references when claimed limitations require a correspondence between items mentioned in different steps. For instance, looking just at the two "parsing" steps, one might assume that one could use any DTD and any source code statement. However, the "selecting" step makes it clear that there must be an association between the source code file and the document definition file. Because one parsing step is being read on Meltzer and the other one on Day, there is neither reason nor suggestion for an association between the source code file and the document definition file. It is submitted that the Examiner has not given a believable explanation why one would combine these two references.

It is further submitted that the Examiner, in his combination of Meltzer and Day is combining the translation of a document with the translation of source code. It is submitted that this is not something that one of ordinary skill in the art would seek to do. This is like combining a German to English translator with a French to English translator - it makes no sense, as they are dealing with different entities. Even if you could combine these different type programs, it is submitted that they would not form the claimed

method. As discussed above, one cannot take the source code from an application and expect to find a correspondence between it and the DTD of a document. This rejection is overcome for Claim 1. Claims 12 and 23 are rejected for similar reasons to Claim 1 and their rejections are overcome for the same reasons.

Independent Claims 5, 16, and 24 are similar to Claims 1, 12, and 23, except that they translate in the opposite direction, from XML to source code. Exemplary Claim 5 recites:

> *5. A method of processing a markup language element, the method comprising the computer-implemented steps of:*
> *parsing a document type definition file for the markup language;*
> *parsing a markup language element from a markup language file;*
> *selecting an element defined in the document type definition file that is equivalent to the markup language element from the markup language file;*
> *generating a source code statement using an identifier of a routine within the selected element; and*
> *writing the source code statement to an output file.*

Although Claims 5, 16, and 24 are not identical to claims 1, 12, and 23, they have many of the same limitations. Both groups require a DTD and they require a correspondence between the DTD and the program to be translated, whether it be a markup language file to be translated to source code or a source code file to be translated to markup language. Meltzer and Day do not provide the necessary relationship between the DTD of Meltzer and the program of Day. It is asserted that these rejections are also overcome.

Claims 2-4 depend from Claim 1 and Claims 13-15 depend from Claim 12. Therefore these claims incorporate the same differences as their parent claims. Additional a number of the dependent claims also recite additional features not disclosed by the cited art.

For instance, Claims 2, 3, 13, and 14 all recite that "the source code statement comprises parameters for the routine".

Claims 4 and 15 recite that "the routine is a procedure, subroutine, function, method, class, or software object."

Therefore, it is urged that the rejections of Claims 2-4 and 13-15 are overcome.

The rejections under 103 have now been overcome.

III. Conclusion

It is respectfully urged that the subject application is patentable over Meltzer *et al.* and Day and is now in condition for allowance.

The examiner is invited to call the undersigned at the below-listed telephone number if in the opinion of the examiner such a telephone conference would expedite or aid the prosecution and examination of this application.

DATE: 9-19-2002

Respectfully submitted,

Betty Formby

Betty Formby
Reg. No. 36,536
Carstens, Yee & Cahoon, LLP
P.O. Box 802334
Dallas, TX 75380
(972) 367-2001
Agent for Applicants

REDACTED SPECIFICATION:

Please amend the specification to read as follows:

On page 1, replace the paragraph of lines 9-14 with the following:

The present application is related to Application Serial Number 09/306,198 [(Attorney Docket Number AT9-98-921)], filed [(concurrently herewith)] 04/30/1999, entitled "Method and Apparatus for Converting Application Programming Interfaces Into Equivalent Markup Language Elements," hereby incorporated by reference.

On pages 12-13, replace the paragraph which extends from page 12, line 15 through page 13, line 1 with the following:

The URL provides a universal, consistent method for finding and accessing this information, not necessarily for the user, but mostly for the user's Web "browser". A browser is a software application for requesting and receiving content from the Internet or World Wide Web. Usually, a browser at a client machine, such as client **308** or data processing system **200**, submits a request for information identified by a URL. Retrieval of information on the Web is generally accomplished with an HTML-compatible browser. The Internet also is widely used to transfer applications to users using a browser[s]. With respect to commerce on the Web, consumers and businesses use the Web to purchase various goods and services. In offering goods and services, some companies offer goods and services solely on the Web while others use the Web to extend their reach. [Information about the World Wide Web can be found at the Web site of the World Wide Web Consortium at http://www.w3.org.]

REDACTED CLAIMS:

1. (Amended) A method of processing a source code statement written in a programming language, the method comprising the computer-implemented steps of:

parsing a document type definition file for a markup language;

parsing [a] said source code statement from a source code file;

selecting an element defined in the document type definition file based on an association between the element and an identifier of a routine in [the] said source code statement; and

writing the selected element to a markup language file.